CLAIMS

1. A computer, comprising:

a processor pipeline designed to alternately execute instructions coded for first and second different computer architectures or coded to implement first and second different processing conventions;

a memory for storing instructions for execution by the processor pipeline, the memory being divided into pages for management by a virtual memory manager, a single address space of the memory having first and second pages;

a memory unit designed to fetch instructions from the memory for execution by the pipeline, and to fetch stored indicator elements associated with respective memory pages of the single address space from which the instructions are to be fetched, each indicator element designed to store an indication of which of two different computer architectures and/or execution conventions under which instruction data of the associated page are to be executed by the processor pipeline;

the memory unit and/or processor pipeline further designed to recognize an execution flow from the first page, whose associated indicator element indicates the first architecture or execution convention, to the second page, whose associated indicator element indicates the first architecture or execution convention, and in response to the recognizing, to adapt a processing mode of the processor pipeline or a storage content of the memory to effect execution of instructions in the architecture and/or under the convention indicated by the indicator element corresponding to the instruction's page.

2. The method of claim 1:

wherein the two architectures are two instruction set architectures;

and wherein the adapting step includes controlling instruction execution hardware of the computer to interpret the instructions according to the two instruction set architectures according to the indicator elements.

3. The method of claim 1, wherein the two conventions are first and second calling conventions, and further comprising:

recognizing when program execution has transferred from a region whose indicator element indicates the first calling convention to a region whose indicator element indicates the second calling convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second calling convention that is logically equivalent to a pre-alteration program context under the first calling convention.

4. A method, comprising:

executing instructions fetched from first and second regions of a single address space of the memory of a computer, the instructions of the first and second regions being coded for execution by computer of first and second architectures or following first and second data storage conventions, respectively, the memory regions having associated first and second indicator elements, the indicator elements each having a value indicating the architecture or data storage convention under which instructions from the associated region are to be executed;

when execution of the instruction data flows from the first region to the second, adapting the computer for execution in the second architecture or convention.

5. The method of claim 4, wherein:

the regions are pages managed by a virtual memory manager.

6. The method of claim 5, wherein the indicator elements are stored in a table of indicator elements distinct from a primary address translation table used by the virtual memory manager, the indicator elements of the table associated with corresponding pages of the memory.

7. The method of claim 5, wherein the indicator elements are stored in a table, each indicator element associated with a corresponding physical page frame.

8. The method of claim 5, wherein the entries are entries of a translation look-aside buffer.

9. The method of claim 4, wherein the regions are lines of an instruction cache.

10. The method of claim 4:

wherein the two architectures are two instruction set architectures;

and wherein the adapting step includes controlling instruction execution hardware of the computer to interpret the instructions according to the two instruction set architectures according to the indicator elements.

11. The method of claim 10, wherein:

the regions are pages managed by a virtual memory manager.

12. The method of claim 11, wherein the indicator elements are stored in a table whose entries are associated with corresponding physical page frames.

13. The method of claim 11, wherein the entry is one entry of a translation look-aside buffer.

14. The method of claim 10, wherein a mode of execution of the instructions is changed without software intervention when execution flows from the first region to the second.

15. The method of claim 10, wherein execution of the computer takes an exception when execution flows from the first region to the second.

16. The method of claim 15, wherein the mode of execution of the instructions is explicitly controlled by an exception handler.

17. The method of claim 10, wherein:

one of the regions stores an off-the-shelf operating system binary coded in an instruction set non-native to the computer, the non-native instruction set providing access to a reduced subset of the resources of the computer.

18. The method of claim 10, wherein the two conventions are first and second data storage conventions, and further comprising:

recognizing when program execution has transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

19. The method of claim 18, wherein:

the regions are pages managed by a virtual memory manager;

one of the two data storage conventions is a register-based calling convention, and the other data storage convention is a memory stack-based calling convention.

20. The method of claim 18, further comprising:

classifying control-flow instructions of a computer instruction set into a plurality of classes; and

during execution of a program on a computer, as part of the execution of instructions of the instruction set, updating a record of the class of the classified control-flow instruction most recently executed;

the adjusting process being determined, at least in part, by the instruction class record.

21. The method of claim 18, wherein:

the instruction data coded for execution by a first of the two instruction set architectures observes a data storage convention associated with the first architecture, and instruction data coded for execution by a second of the two instruction set architectures observes a second, different, data storage convention associated with the second architecture, a single indicator element indicating both the instruction set architecture and the data storage convention;

and further comprising, recognizing when program execution transfers from a region using the first instruction set architecture to a region using the second instruction set architecture, and in response to the recognition, adjusting the data storage content of the computer from the first storage convention to the second..

22. The method of claim 4, wherein the two conventions are first and second data storage conventions, and further comprising:

recognizing when program execution has transferred from a region whose indicator element indicates the first data storage convention to a region whose indictor element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

23. The method of claim 22, further comprising:

overlaying the logical resources of the first and second instruction set architectures onto the physical resources of the computer according to a mapping that assigns corresponding resources of the two architectures to a common physical resource of a computer when the resources serve analogous functions in the calling conventions of the two architectures.

24. The method of claim 22, wherein the adjusting step further comprises:

altering a bit representation of a datum from a first representation in the first convention to a second representation in the second convention, the alteration of representation being chosen to preserve the meaning of the datum across the change in execution convention.

25. The method of claim 22, wherein the adjusting step further comprises:

copying a datum from a first location to a second location, the first location having a use under the first data storage convention analogous to the use of the second location under the second data storage convention.

26. The method of claim 22, wherein the adjusting step further comprises:

copying a datum from a third location to a fourth, the third location having a use under the first data storage convention analogous to the use of the third location under the first data storage convention and to the fourth location under the second data storage convention, a program for the copying being programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program.

27. The method of claim 22, wherein

a rule for copying data from the first location to the second is determined by examining a descriptor associated with the location of execution before the recognized execution transfer.

28. The method of claim 22, wherein the two conventions are two calling conventions.

29. The method of claim 28, wherein:

one of the two calling conventions is a register-based calling convention, and the other calling convention is a memory stack-based calling convention.

30. The method of claim 28, wherein:

the regions are pages managed by a virtual memory manager.

31. The method of claim 30, wherein the indicator elements are stored in a table whose entries are associated with corresponding physical page frames.

32. The method of claim 31, wherein the entry is one entry of a translation look-aside buffer.

33. The method of claim 28, further comprising:

taking a processor exception in response to the recognition, a handler for the exception programmed to copy a datum from a first location to a second location, the first location having a use under the first data storage convention analogous to the use of the second location under the second data storage convention.

34. The method of claim 22, further comprising:

classifying control-flow instructions of a computer instruction set into a plurality of classes; and

during execution of a program on a computer, as part of the execution of instructions of the instruction set, updating a record of the class of the classified control-flow instruction most recently executed;

the adjusting process being determined, at least in part, by the instruction class record.

35. The method of claim 34, wherein:

one of the two data storage conventions is a register-based calling convention, and the other data storage convention is a memory stack-based calling convention.

36. The method of claim 34, wherein:

in some of the control-flow instructions, the classification is statically determined by the opcode of the instructions; and

in other of the control-flow instructions, the classification is dynamically determined based on a full/empty status of a register.

37. A computer processor, comprising:

a processor pipeline configured to alternately execute instructions of computers of two different architectures or processing conventions;

a memory unit designed to fetch instructions from a computer memory for execution by the pipeline, and to fetch stored indicator elements associated with respective memory regions of a single address space from which the instructions are to be fetched, each indicator element designed to store an indication of the architecture or execution convention under which the instruction data of the associated region are to be executed by the processor pipeline;

the memory unit and/or processor pipeline further designed to recognize an execution flow from a region whose indicator element indicates one architecture or execution convention to another.

38. The method of claim 37, wherein the indicator elements are stored in a table of indicator elements distinct from a primary address translation table used by the virtual memory manager, the indicator elements of the table associated with corresponding pages of the memory.

39. The computer processor of claim 37, further comprising:

a translation look-aside buffer (TLB); and

TLB control circuitry designed to load the indicator elements into the TLB from a table stored in memory, the entries of the table being associated with corresponding physical page frames.

40. The computer processor of claim 37, wherein the two architectures are two instruction set architectures, and further comprising:

processor pipeline control circuitry designed to control the processor pipeline to effect interpretation of the instructions under the two instruction set architectures alternately, according to the associated indicator elements.

41. The computer processor of claim 40, further comprising:

software programmed to manage a transition between the execution of a program executing in the first instruction set architecture, being an instruction set architecture native to the computer processor, and execution of an off-the-shelf operating system coded in the second instruction set, being an instruction set non-native to the computer, providing access to a reduced subset of the resources of the computer.

42. The computer processor of claim 40:

each indicator element being further designed to store an indication of a calling convention under which the instruction data of the associated region are coded for execution by the processor pipeline;

and further comprising software programmed to alter the data storage content of a computer using the computer processor to create a program context under the second calling convention that is logically equivalent to a pre-alteration program context under the first calling convention;

the memory unit further designed to recognize when program execution has transferred from a region whose indicator element indicates the first calling convention to a region whose indicator element indicates the second calling convention, and in response to the recognition, to invoke the transition management software.

43. The computer processor of claim 42, wherein the memory unit is designed to recognize a single indicator element to indicate both the instruction set architecture and calling convention of a region.

44. The computer processor of claim 42, wherein the memory unit and software are designed to effect a transition between instruction boundaries, between execution in a region coded in the first instruction set using the first calling convention to execution in a region coded in the second instruction set using the second calling convention, so that code at the source of the transfer may effect the execution transition without being specially coded for code at the destination.

45. The computer processor of claim 42, wherein:

one of the two calling conventions is a register-based calling convention, and the other calling convention is a memory stack-based calling convention.

46. The computer processor of claim 42, wherein the logical resources for support of the first and second instruction set architectures are overlaid on the physical resources of the computer processor according to a mapping that assigns corresponding resources of the two architectures to a common physical resource when the resources serve analogous functions in the calling conventions of the two architectures.

47. The computer processor of claim 42, wherein

a rule for altering the data storage content from the first calling convention to the second is determined by examining a descriptor associated with the location of execution before the recognized execution transfer.

48. The computer processor of claim 42, wherein control-flow instructions of the instruction set are classified into a plurality of classes; and

the processor pipeline updates a record of the class of the classified control-flow instruction most recently executed;

the storage alteration process being determined, at least in part, by the instruction class record.

49. The computer processor of claim 40, further comprising:

a transition manager designed to effect a transition between the execution of code coded in instructions of a first instruction set architecture and code coded in instructions of a second instruction set architecture, the transition manager designed to alter a bit representation of a datum from a first representation under the first architecture to a second representation under the second architecture, the alteration of representation being chosen to preserve the meaning of the datum across the change in execution architecture.

50. The computer processor of claim 40, further comprising circuitry designed to raise an exception when the execution flow from a region whose indicator element indicates one architecture or execution convention to another is recognized.

51. A method, comprising:

storing instructions in pages of a computer memory managed by a virtual memory manager, the instruction data of the pages being coded for execution by, respectively, computers of two different architectures and/or under two different execution conventions;

in association with pages of the memory, storing corresponding indicator elements indicating the architecture or convention in which the instructions of the pages are to be executed;

executing instructions from the pages in a common processor, the processor designed, responsive to the page indicator elements, to execute instructions in the architecture or under the convention indicated by the indicator element corresponding to the instruction's page.

52. The method of claim 51, wherein the pages' indicator elements are stored in a table whose entries are associated with corresponding physical page frames, and cached in a translation look-aside buffer.

53. The method of claim 51, wherein the two architectures are two instruction set architectures, and further comprising:

controlling the instruction execution hardware of the computer to interpret the instructions according to the two instruction set architectures according to the indicator element.

54. The method of claim 51, wherein the two conventions are first and second data storage conventions, and further comprising:

recognizing when program execution has transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention.

55. The method of claim 54, further comprising:

storing instruction data in a third page, the instruction data of the third page being coded for execution by one of the two architectures, and observing a data storage convention associated with the other of the two architectures;

storing indicator elements indicating the data storage convention observed by the instructions of the respective pages; and

recognizing each transition of program execution from a page using the first data storage convention to a page using the second data storage convention, and in response to the recognition, adjusting the data storage content of the computer from the first storage convention to the second, and vice-versa.

56. The method of claim 53, wherein:

the instruction data coded for execution by a first of the two instruction set architectures observes a data storage convention associated with the first architecture, and instruction data coded for execution by a second of the two instruction set architectures observes a second, different, data storage convention associated with the second architecture, a single indicator

element indicating both the instruction set architecture and the data storage convention of the associated page;

and further comprising, recognizing when program execution transfers from a page using the first instruction set architecture to a page using the second instruction set architecture, and in response to the recognition, adjusting the data storage content of the computer from the first storage convention to the second.

57. The method of claim 54, wherein the two conventions are a register-based calling convention and a memory stack-based calling convention, and further comprising:

recognizing when program execution has transferred from a page using the register-based calling convention to a page using the memory stack-based convention, and in response to the recognition, adjusting the data storage content of the computer from the first calling convention to the second.

58. The method of claim 54, wherein the adjusting step further comprises:

copying a datum from a third location to a fourth, the third location having a use under the first data storage convention analogous to the use of the third location under the first data storage convention and to the fourth location under the second data storage convention, a program for the copying being programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program, a bit representation of the datum copied to the second location differing from a bit representation of the datum copied from the first location, the alteration of representation being chosen to preserve the meaning of the datum across the change in data storage convention.

59. The method of claim 54, wherein

a rule for copying data from the first location to the second is determined by examining a descriptor associated with the location of execution before the recognized execution transfer.

60. The method of claim 54, further comprising:

classifying control-flow instructions of a computer instruction set into a plurality of classes; and

during execution of a program on a computer, as part of the execution of instructions of the instruction set, updating a record of the class of the classified control-flow instruction most recently executed;

the altering process being determined, at least in part, by the instruction class record.

61. A microprocessor chip, comprising:

an instruction unit, configured to fetch instructions from a memory managed by the virtual memory manager, and configured to execute instructions coded for first and second different computer architectures or coded to implement first and second different data storage conventions;

the microprocessor chip being designed (a) to retrieve indicator elements stored in association with respective pages of the memory, each indicator element indicating the architecture or convention in which the instructions of the page are to be executed, and (b) to recognize when instruction execution has flowed from a page of the first architecture or convention to a page of the second, as indicted by the respective associated indicator elements, and (c) to alter a processing mode of the instruction unit or a storage content of the memory to effect execution of instructions in accord with the indicator element associated with the page of the second architecture or convention.

62. The method of claim 61, wherein the indicator elements are stored in virtual address translation table entries.

63. The method of claim 61, wherein the indicator elements are stored in a table distinct from a primary address translation table used by a virtual memory manager, the indicator elements of the table being stored in association with respective pages of the memory.

64. The method of claim 61, wherein the indicator elements are stored in association with respective physical page frames.

65. The method of claim 61, wherein the indicator elements are stored in association with respective virtual pages.

66. The method of claim 61, wherein the indicator elements are stored in entries of a translation look-aside buffer.

67. The method of claim 61, wherein the indicator elements are stored in an instruction cache.

68. The microprocessor chip of claim 61, wherein a mode of execution of the instructions is changed without software intervention when execution flows from the first region to the second.

69. The microprocessor chip of claim 61, the microprocessor chip being designed to raise an exception when execution flows from the first region to the second;

and further comprising exception handler software programmed to handle the exception by explicitly controlling a mode of execution of the instructions.

70. The microprocessor chip of claim 61, wherein the architecture or convention indicator elements are stored in a table whose entries are associated with corresponding physical page frames, and cached in a translation look-aside buffer.

71. The microprocessor chip of claim 61, wherein the two architectures are two instruction set architectures, and the microprocessor chip controls the instruction unit to interpret the instructions according to the two instruction set architectures according to the indicator element corresponding to the pages from which the instructions are fetched.

72. The microprocessor chip of claim 71, further comprising:

software programmed to manage a transition between the execution of a program executing in the first instruction set architecture, being an instruction set architecture native to the computer processor, and execution of an off-the-shelf operating system coded in the second instruction set, being an instruction set non-native to the computer providing access to a reduced subset of the resources of the computer.

73 The microprocessor chip of claim 71:

each indicator element being further designed to store an indication of a data storage convention under which the instruction data of the associated page are coded for execution by the instruction unit;

and further comprising software programmed to alter the data storage content of a computer using the microprocessor chip, to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention;

the microprocessor chip further designed to recognize when program execution has transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, to invoke the transition management software.

74. The microprocessor chip of claim 73, being further designed to:

to retrieve instruction data from a third page, the instruction data of the third page being coded for execution by a first of the two architectures, and observing a data storage convention of the second of the two architectures;

to retrieve indicator elements indicating the data storage convention observed by the instructions of the respective pages; and

to recognize each transition of program execution from a page using the first data storage convention to a page using the second data storage convention, and in response to the recognition, to adjust the data storage content of the computer from the first data storage convention to the second data storage convention, and vice-versa.

75. The microprocessor chip of claim 73, further designed to recognize a single indicator element to indicate both the instruction set architecture and calling convention of a page.

76. The microprocessor chip of claim 71, further comprising hardware and/or software designed:

(a) to retrieve calling convention indicator elements stored in association with respective pages of the memory, each calling convention indicator element indicating which of a register-based calling convention or a memory stack-based calling convention is observed by instructions of the page;

(b) to recognize when instruction execution has flowed from a page of a memory-based convention to a page of the register-based calling convention, as indicated by the calling convention indicator elements associated with the respective pages, and

(c) in response to the recognition, to alter a storage content of the computer to create a program context under the register-based convention logically equivalent to a pre-alteration program context under the memory-based convention.

77. The microprocessor chip of claim 61:

wherein the two conventions are first and second data storage conventions;

and further comprising software programmed to alter the data storage content of a computer using the microprocessor chip, to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention;

the microprocessor chip being further designed to recognize when program execution has transferred from a region whose indicator element indicates the first data storage convention to a region whose indicator element indicates the second data storage convention, and in response to the recognition, to invoke the transition management software.

78. The microprocessor chip of claim 77, wherein the microprocessor chip and software are designed to effect a transition between instruction boundaries, between execution on a page coded in the first instruction set using the first calling convention to execution on a page coded in the second instruction set using the second calling convention, the software programmed to effect the execution transition without the code at the source of the transition being specially coded to interface with code at the destination of the transition.

79. The microprocessor chip of claim 77, wherein the two conventions are two calling conventions.

80. The microprocessor chip of claim 79, wherein:

one of the two calling conventions is a register-based calling convention, and the other calling convention is a memory stack-based calling convention.

81. The microprocessor chip of claim 79, wherein the physical resources of the microprocessor chip are associated to the logical resources of the first and second calling conventions according to a mapping that assigns corresponding logical resources to a common physical resource when the resources serve analogous functions in the two calling conventions.

82. The microprocessor chip of claim 79, further comprising:

software and/or hardware designed to effect a transition between the execution of code coded under the first calling convention and code coded under the second calling convention, by altering a bit representation of a datum from a first representation under the first calling convention to a second representation under the second calling convention, the alteration of representation being chosen to preserve the meaning of the datum across the change in calling convention.

83. The microprocessor chip of claim 79, further comprising:

software and/or hardware designed to copy a datum from a first location to a second location, the first location having a use under the first calling convention analogous to the use of the second location under the second calling convention.

84. The microprocessor chip of claim 79, further comprising:

software and/or hardware designed to copy a datum from a third location to a fourth, the third location having a use under the first data storage convention analogous to the use of the third location under the first data storage convention and to the fourth location under the second data storage convention, the software and/or hardware for the copying being programmed to assume that exactly one of the first and third locations is no longer required by the execution of the program..

85. The microprocessor chip of claim 61, further comprising hardware and/or software designed:

(a) to retrieve calling convention indicator elements stored in association with respective pages of the memory, each calling convention indicator element indicating which of a register-based calling convention or a memory stack-based calling convention is observed by instructions of the page;

(b) to recognize when instruction execution has flowed from a page using the register-based calling convention to a page using the memory stack-based convention, as indicated by the calling convention indicator elements associated with the respective pages, and

(c) in response to the recognition, to alter a storage content of the computer to create a program context under the memory-based convention logically equivalent to a pre-alteration program context under the register-based convention.

86. The microprocessor chip of claim 61, wherein control-flow instructions of the microprocessor's instruction set are classified into a plurality of classes; and

the fetch and execute unit updates a record of the class of the classified control-flow instruction most recently executed;

the storage alteration process being determined, at least in part, by the instruction class record.

87. A method, comprising:

executing a control-transfer instruction under a first execution context of a computer, the instruction being architecturally defined to transfer control directly to a destination instruction for execution in a second execution context of the computer;

before executing the destination instruction, altering the data storage content of the computer to establish a program context under the second execution context that is logically equivalent to the context of the computer as interpreted under the first execution context, the reconfiguring including at least one data movement operation not included in the architectural definition of the control-transfer instruction.

copying data from a general register to a memory stack; and

copying data from a memory stack to a general register.

88. The method of claim 87, wherein the data movement includes at least one of:

copying data from a general register to a memory stack; and

copying data from a memory stack to a general register.

89. The method of claim 87, wherein:

the transition for the first execution context to the second context is recognized in a difference between two indicator elements associated with the memory pages containing the control-transfer instruction and the destination instruction, respectively, the pages being under the management of a virtual memory manager.

90. The method of claim 89, wherein the indicator elements are stored in a table whose entries are associated with corresponding physical page frames.

91. The method of claim 89, wherein the indicator elements are stored in entries of a translation look-aside buffer.

92. The method of claim 87, further comprising:

altering a bit representation of a datum from a first representation in the first convention to a second representation in the second convention, the alteration of representation being chosen to preserve the meaning of the datum across the change in execution convention.

93. The method of claim 87, wherein

a rule for copying data from the source memory or register to the destination register or memory is determined by examining a descriptor associated with the location of the control-transfer instruction.

94. A method, comprising:

executing a section of computer object code twice, without modification of the code section between the two executions, the code section materializing a destination address into a register and being architecturally defined to directly transfer control indirectly through the

register to the destination address, the two executions materializing two different destination addresses;

the two destination code sections at the two materialized destination addresses being coded in two distinct instruction sets, neither instruction set being a subset of the other.

95. The method of claim 94, wherein:

the code at the first destination receives floating-point arguments and returns floating-point return values using a register-based calling convention; and

the code at the second destination receives floating-point arguments using a memory-based stack calling convention, and returns floating-point values using a register indicated by a top-of-stack pointer.